Chair of Automation and Information Systems
Department of Mechanical Engineering
Technical University of Munich

Modeling and Analyzing Communication Delays in Multi-Agent System with Two Industry Use Cases

Modellierung und Analyse von Kommunikationsverzögerungen in Multi-Agent-Systemen mit zwei Anwendungsfällen

Semester Thesis

at the Department of Mechanical Engineering of the Technical University of Munich

Supervised by Prof. Dr.-Ing. Birgit Vogel-Heuser
M.Sc. Jingyun Zhao
M. Sc. Fandi Bi
Chair of Automation and Information Systems

Submitted by Xuezhou Hou
Willi-Graf-Straße 5
80805 München

Submitted on November 15, 2023 in Garching

Scope of Work

Title of the Master's Thesis:
Manually Change you Thesis Title here

Author:	B.Sc. Max Mustermann	**Supervisor:**	M.Sc. Mein Betreuer
Issuance:	01.07.2021	**Submission:**	31.12.2021

Setting:
Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Objective:
Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Methodology:
The content of the present thesis can be subdivided into the following tasks

- Method 1
- Method 2
- Method 3
- etc.

Declaration

I hereby confirm that this master's/ bachelor's/ semester thesis was written independently by myself without the use of any sources beyond those cited, and all passages and ideas taken from other sources are cited accordingly.

.... Location, Date

.... Signature

With the supervision of Mr. Max Mustermann by Mr. Mein Betreuer intellectual property of the Professorship Laser-based Additive Manufacturing (LBAM) flows into this work. A publication of the work or a passing on to third parties requires the permission of the head of the professorship. I agree to the archiving of the printed thesis in the LBAM library (which is only accessible to LBAM staff) and in LBAM's digital thesis database as a PDF document.

.... Location, Date

.... Signature

Abstract

Here could be your abstract.

Zusammenfassung

Hier könnte Ihre Kurzzusammenfassung stehen.

Contents

1 Introduction

1.1 Motivation

The requirements of modern industrial production has become more and more crucial, due to the increased complexity of interconnections of various different robots and automation systems during manufacturing. Different from the production mode of traditional factory, the concept of smart factory of Industry 4.0 has been developed to overcome the remaining issues such as high centralization of traditional control system, low scalability of production systems and processes, limited adaptability of new conditions and requirements with changing environments, hardness of real-time decision making, insufficient resource allocation and many more. Under this prerequisite, a Multi-Agent System (MAS) plays a crucial part to close the gaps.

The precursor to modern MAS is Distributed Artificial Intelligence (DAI), which focuses on distributing a single complex AI task to multiple machines and processors[4]. The concept of resource distribution/decentralization is inherited by MAS to handle more distributed and interconnected computing tasks and results in a more intelligent and autonomous agent based operation system, in which each agent can make decision independently to achieve its own goal while still tend to collaborate, negotiate and coordinate with other agents frequently, in order to improve the efficiency and quality of production workflow meanwhile reducing cost [8]. The capability of agents being able to interact with each other, perceive and adapt to the rapid changes in the environment makes it possible for MAS to solve comprehensive problems which a single agent cannot.

1.2 Research questions

RQ1: How can agents communicate with each other in MAS?

RQ2: How to measure the delays of data exchanges between different Resource Agent (RA) and delays of data upload/download within Digital Twin (DT) agent?

RQ3: How to modularize the delays with Domain Specific Language (DSL)?

1.3 Outline

In this thesis, chapter 2 introduce the concepts and utilization of MAS, the Network communication principles and protocols, DSL for Cyber Physical System (CPS) with the concentration of robotics, and DT. Chapter 3 summarizes the state of art of all those concepts in chapter 2. After that, chapter 4 comes up with methodologies of building a MAS for communication and a DT agent for data update and chapter 5 comes up with the implementation results of different test cases and two use cases for performance testing on the MAS and DT agent.

2 State of the art

2.1 MAS

In order to adapt the requirements a general smart factory, a more detailed MAS should be designed and chosen carefully at the beginning. Within the subfields of MAS, an agent can be identified as a software agent that has no physical embodiment but only software to control physical assets for different purposes. "In agent-oriented software development, an agent is the concept of a delineable software unit with a defined goal. An agent tries to achieve this goal through autonomous behavior, continuously interacting with its environment and other agents." [9] However, although not under discussion in this thesis, other interpretation of MAS can be a Multi-agent robot systems that each agent represents an actual physical objects such as an individual robot that participates in the complex task execution. [7] Different researches define MAS from different aspects, which can be confusing for problems clarity. To simplify the concept, a general MAS can be subdivided into three views: the technical system that comprises the robots, the automation control system that is characterized by sensors, actuators, networks and robot control units, and the technical process that describes the product's production process.[5] [10]The focus of this thesis is mainly in technical system which can be interpreted as a union of robots in a smart factory, and the components of a robot or functions executed for a specific movement is less under concern here.

For further discretization of an agent, whether the agent is product, process or resource oriented, an appropriate agent architecture should be chosen according to different considerations. There are several of them should be emphasized: RA, Communication Agent (CA), and Agent Management System (AMS). RA is agent in field level representing a single robot. Different from then other agents, RA should be able to combine the modules with physical entities by choosing an appropriate design pattern. Therefore, a comparison of design patterns in different production levels is done [6]. The choice of an ideal design pattern should be limited for RA in this research by comparing three relevant design patterns: RA pattern in Wannagat's architecture, Material Flow System (MFS) patterns in Fischer's architecture self*-control MAS in Ryashentseva's architecture. Among all, Wannagat's architecture is chosen as the appropriate design patterns for RA for field level control, which consists of four modules: Planning Module, Knowledge Base, Control Module and Diagnosis Module. All modules are interconnected meanwhile with each connected to I/Os of physical system and a communication interface to interact with other Resource Agents (RAs) or AMS through CA [1]. AMS and CA on the other hand should have different specifications in the same design pattern. CA for example, should be able to coordinate the message-based communication between the agents, as a "mailbox" between them while AMS plays an important role in centralization and coordination of all other agents [11].

2.2 Network communication

2.3 DSL

2.4 Digital Twin

Apart from the agents that are responsible for internal production processes, Digital Twin agent plays a crucial part in collecting data from the production and storing it externally to create a digital replica of the physical entities or systems. The concept of DT was first introduced by Michael Grieves [2], who also introduce the famous Product Lifecycle Management (PLM) conceptual diagram [3]to explain the role of DT in the product lifecycle. Data from engineering, design and manufacture should be digitalized to represent physical assets. Some common understandings of DT of engineering data could be for example simulation for performance testing. Or may be design data that builds a visual representation of CAD data of plants and robots, and manufacture data that helps to inspect changes of production for process optimization. However, none of these combines all data from PLM model for a digital overview of the whole factory. They can be described as local DT, which should be extended to the concept of global DT, by summarizing the data in a cloud platform and visualizing it with a real-time 3D graphics collaboration platform.

With all these different types of agents following the same workflow, it is fascinating to investigate how data exchanges can be realized in a real-time behavior. Which is the main topic throughout the whole thesis.

2.5 Research gap

3 Methodology

3.1 Internal

3.1.1 Overview (conceptual diagram)

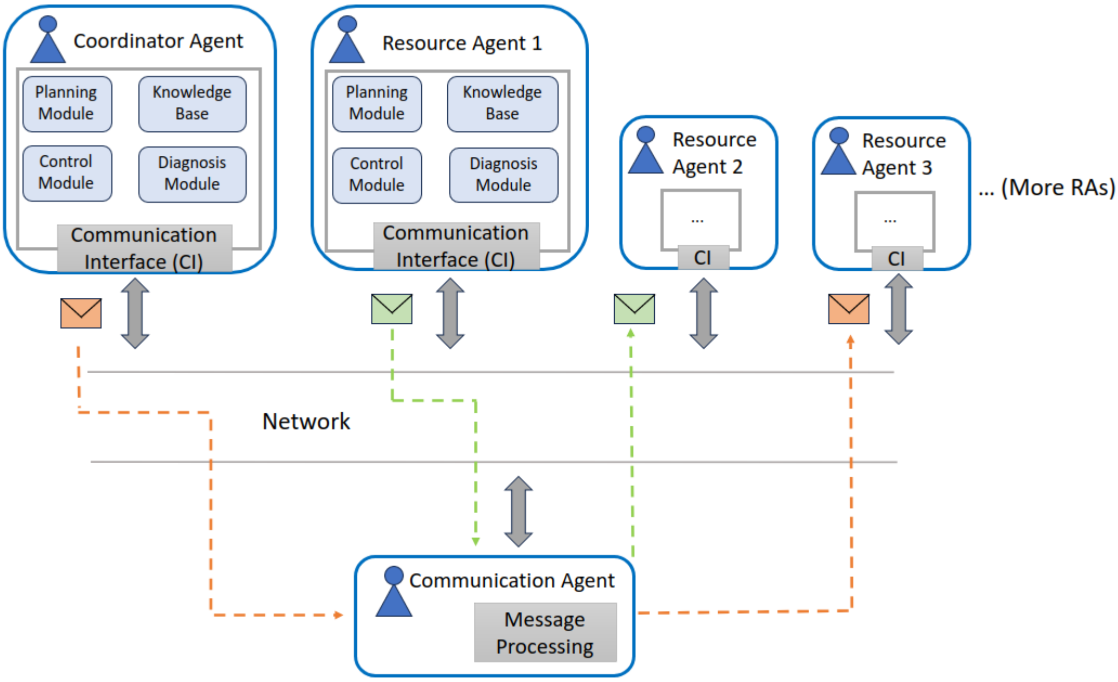

Figure 1: Conceptual diagram of MAS

In the figure.1 shows a conceptual diagram of a MAS based on the RA design patterns in Wannagat's architecture, with the focus on communication between agents, planning and decision making inside each agent. The Coordinator Agent (CDA) here is identical to AMS of Wanagat, which should also be counted as an agent instead of a management system. The five modules within an agent are:

- Planning Module
- Control Module
- Knowledge Base
- Diagnosis Module
- Communication Interface.

Each module within a Coordinator agent or a RA Based on these five modules, the task to be executed in an agent should also be categorized into 5 parts. The following table shows some tasks of each module based on the general requirements of a smart factory.

Table 1: Wanagat's RA design patterns with task related examples.

Wanagat's design patterns		
Module name	Task	Example
Planning Module	• Task planning • Decision making • Resource allocation • Sequencing • Scheduling	• Break down tasks into smaller executable units • Decide which task should be assigned to which agent • Allocate the agents with specific tasks • Find the task execution sequence • Calculate the execution time for each agent
Control Module	• Monitoring • Adaptation • Control and optimization • Resource allocation • Actuation	• Acquisition of robot states • Adapt the plans with current state, e.g.: emergent stop • Control and optimize the robot's motion • Allocate the agents with specific tasks • Actuate the robot with outputs
Knowledge Base	• Database (DB) • Knowledge representation and reasoning • Learning • Knowledge sharing	• Hierarchical, relational, non-relational and object oriented • Relational ontology DB system • Agent learns from the existing primitives and create new executable primitives for customer's changing requirements • Unfound primitives could be retrieved by querying other agents
Diagnosis Module	• Fault detection • Fault diagnosis • Root cause analysis and classification • Fault prediction	• Monitor the time-series data to detect anomalies from robot states, e.g.: detect faulty joint values for abortion • More complex analysis to diagnose faulty patterns with mathematical algorithms and models, or AI-based methods • Find the reasons for anomalies and categorize them for patterns recognition • predict the system faults by learning the classification models
Communication Interface	• Message parsing and encoding • Connection establishment and maintenance • Message handling • Data security	• Encode and decode the messages in agent specific data type, or parse the data object to other types, e.g.: json • Ensure the connection with other agents based on system requirements • Filter messages with undesired data type or incomplete messages, and prioritize the incoming messages • Ensure data integrity and confidentiality, by encrypting, decrypting and authenticating messages to avoid cyber attack

3.1.2 Prerequisite

System Setup

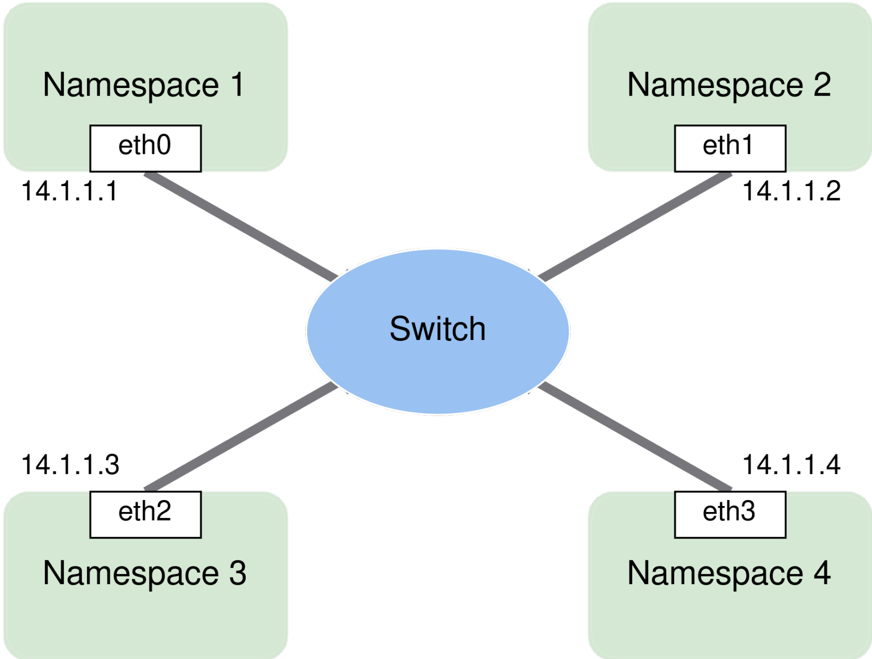

Figure 2: Conceptual diagram of namespaces creation

In order to emulate network environments for agent's communication testing and development of the MAS, the internal packets routing between agents in a single Linux device should be avoided. A common way to visualize the network to do performance testing is to use namespaces for network emulation. The trick is that a process running within a given namespace will see only the network interfaces, including virtual interfaces, forwarding tables, etc., that exist in that namespace. The applications under test should serve as a switch and each packet should be routed through these interfaces. The Figure.2 shows that, each namespace is assigned with a virtual ethernet interface, starting with the name eth, which is configured with an individual IP address. Each time a script gets called, it is running under a namespace with its own IP address. In exercise, if a packet is sent from Namespace 1 to Namespace 3 and then back, it is routed by the switch intead of bridges between namespaces, which are not configured here in order to avoid internal routing.

3.1.3 OSI model and comparison between sockets relevant protocol layers

Figure. 3 shows the famous OSI-Model with 7 abstraction layers with Transport layer and application layers most relevant to sockets. TCP and UDP are typical transport layer protocols and they provide an end-to-end data transport between two devices while the application layer protocols like HTTP or Websocket establish a communication between applications within devices. Although the application layer protocols still utilize TCP/UDP sockets to transport stream data, they defined additional "rules" to specify structure, content, and semantics of the messages transport through sockets. In the following tables, a comparison between protocols in different layers provide a more straight forward overview of their pro and cons in different contents.

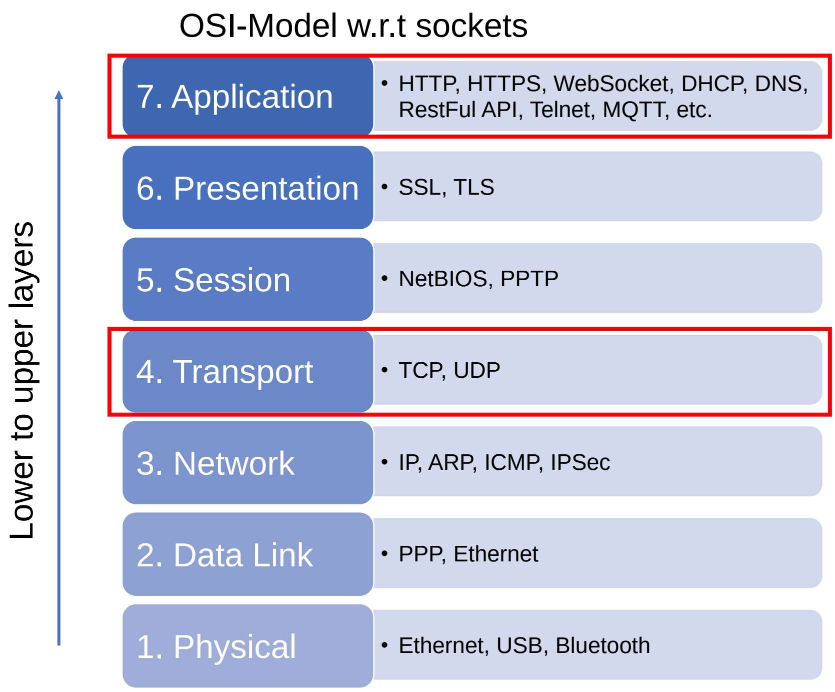

Figure 3: Conceptual diagram of Namespaces creation

Transport layer protocols

Table 2 compares the typical transport layer protocols Transmission Control Protocol (TCP) and User Diagram Protocol (UDP) from different aspects. TCP provides reliable data transfer while UDP mainly focuses on transport speed and efficiency without reliability guarantee. With the focus on speed and reliability, UDP on one hand offers a faster packet transfer, while on the other hand produces a network dependent packet loss rate and out-of-order packet sequence, compared to TCP. Because of the requirement of a reliable real time ordered data transfer with minimum to no packet loss in MAS communication, TCP is used as the base protocols of the design.

Application layer protocols

There are several protocols in application layers that are considered to be suitable for MAS communication, each with its own advantages and limits in different aspects according to table 3. To get a closer look of the Table 3, a horizontal comparisons between different application layer protocols:

- Hypertext Transfer Protocol (HTTP),
- WebSocket,
- RESTful API,
- and Message Queuing Telemetry Transport (MQTT),

should be performed. In one word, WebSocket is chosen to be the application layer protocols for MAS communication, while TCP socket for the DT agent design, which will be further discussed in the next chapter. Here are the reasons of the choice of WebSocket:

Table 2: Characteristics of different technologies in transportlayer protocols

Transport layer protocols		
Aspect	**TCP**	**UDP**
Use cases	Web browsing, email, text messaging, and file transfers	Live and real-time data transmission
Reliability	Reliable	Unreliable
Stream type	Byte stream with no preserved boundaries	Message stream with preserved boundaries
Connection type	Connection oriented, three handshake	No connection needed
Overhead	Larger than UDP	Very low
Header size	20-60 bytes	8 bytes
Sequence	Packets arrive in sequence	No sequencing for packets
Retransmission of lost packets	Yes	No
Speed	Slower than UDP, because of overhead and connection	Relative faster than TCP
State	Stateful	Stateless
Flow control	Yes	No

1. Bi-directional, full duplex, real time communication between server and clients, no re-connection needed. Suitable for continuous data transfer.

2. Overhead small after connection establishment to reduce latency(HTTP).

3. Stateful, store the information of the client's state under connection.

4. High flexibility, adaptability and scalability

5. Secure with wss.

Figure.4 shows the differences between bi-directional full duplex and other message patterns like Request-Response and Publish-Subscribe, in the context of data transfer. For the MQTT, publisher publish (send) a message within a topic to the broker (server), while subscriber subscribes (receive) the message from the broker within the same topic. For response, a new topic needed to be started, but there is no guarantee that the original publisher is listening, which is a drawback for send-and-receive patterns of MAS.

Relatively, the other three protocols will be considered as more appropriate. For instance both HTTP and RESTful API run in request-response mechanism. A client posts (sends) messages to the server for the other client will get (receive) from it. Whether the GET and POST method are successful or not, a response will be given back. With this mechanism, each time a message is going through the server, a new connection will be established and closed after responses are sent. The inconsistent connection will consume more communication time and lead to higher latency. The ideal solution for that is the bi-directional and full duplex real time communication mechanism of WebSocket. Basically the client sends a message to the server, after processing the data, the server will pass the message to the other client and receive response with the same logic. This allows simultaneous communication in both directions between clients, so that no re-connection in the message transfer cycle is necessary. The consistent server-client connection makes it possible to realize a continuous real time communication between agents. In the next section, a more detailed explanation of WebSocket mechanism for MAS design will be given.

Table 3: Characteristics of different technologies in application layer protocols

Application layer protocols				
Aspect	**HTTP**	**WebSocket**	**RESTful API**	**MQTT**
Use cases	Web pages, images, videos, World Wide Web, etc	Such as chat applications, live gaming, etc	Web and mobile applications with data management requirement	Usually in IoT with limited bandwidth
Functionality	Request-response protocol based on TCP, foundation for both RESTful APIs and the initial connection in WebSockets	Bi-directional, real-time communication	Uses standard HTTP methods to perform CRUD (Create, Read, Update, Delete)	Lightweight message transport, runs over TCP
Security	Use SSL/TLS	ws (unsecured) and wss (secured with SSL/TLS)	Similar to HTTP, can be further secured using various authentication mechanisms like OAuth, JWT	Use TLS, like username/password authentication and optional message-level security
Message patterns	Request-Response	Full Duplex (send and receive independent)	Request-Response	Publish-Subscribe
Connection type	No connection needed	Persistent connection	No connection needed	Persistent connection
State	Stateless	Stateful	Stateless	Stateful
Overhead	Overhead for each request-response cycle, especially for new connections	After the initial handshake (HTTP), data frames are lightweight	Similar to HTTP, dependent on API design	Minimal message overhead
Realtime capability	Less Capable	Highly Suitable	Variable	Highly Suitable
Flexibility	Supported in all environments	Supported in most modern web browsers and many backend environments	Similar to HTTP	Highly flexible
Adaptability to dynamic changes	Relative lower (influenced by stateless nature)	High	Relative lower (influenced by stateless nature)	High
Capability of handling instability	Less capable, requires a stable connection for each request-response cycle	Less stable if connection disruptions happen frequently	Identical to HTTP	Capable, Ideal for remote locations with limited connectivity
Scalability	Less scalable, require more infrastructure support	Highly scalable, maintains connections for real-time interactions	similar to HTTP, dependent on API design	Highly scalable based on broker-client message transport

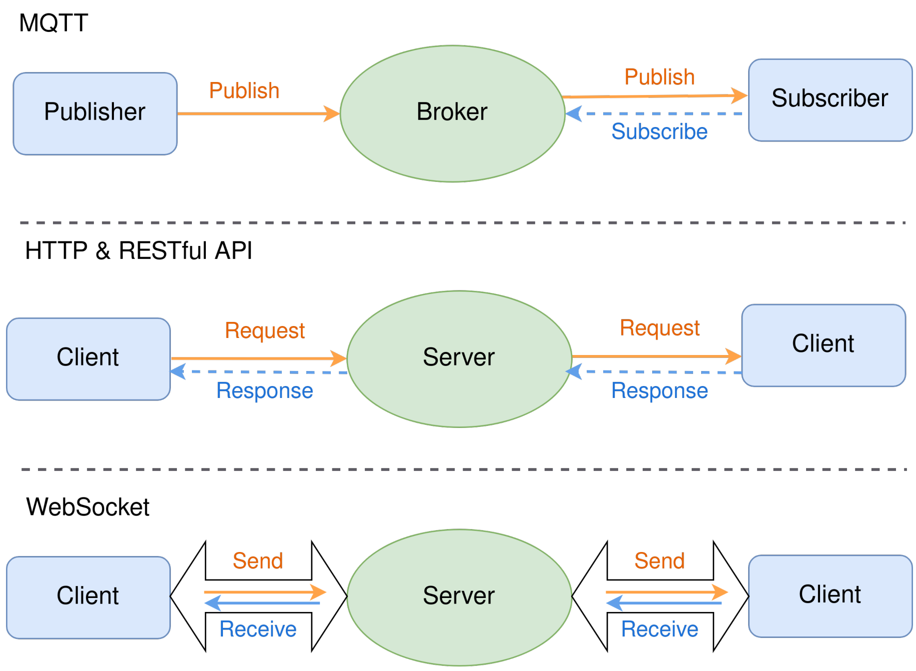

Figure 4: Conceptual diagram of MAS

3.1.4 Transport layer protocols versus Application layer protocols

The reasons of choosing application layer protocols over transport layer protocols for MAS according to table 2 and 3 are listed as following:

1. Semantic-rich message patterns.
2. Higher flexibility.
3. Standardized communication between different agents.
4. Header with agent relevant information.
5. Security and authentication.

However, unlike for MAS design, the architecture of DT agent is more straightforward and the basic TCP socket is chosen for the more to one server-client communication. The details will be discussed later.

3.1.5 Pseudo-Code of MAS workflow in WebSocket

In Algorithm 1 is a piece of pseudocode that reflects the workflow of the MAS for the CDA. Unlike other RAs, CDA has no mechanisms of primitives execution but the ability to do planning and decision making. Under the Main, the production tasks are broken down into a set of sequenced primitives and then assigened to each allocated agent through one of the predefinded *send_and_receive_messages* functions under the messageSender class. After getting responses from the agents, CDA should deicide whether to start processing by positive responses or retry the steps by negative responses. Once the process starts, the CDA does nothing but wait for the inform messages from the allocated agents by using one of the *receive_and_send_messages* functions under the messageReceiver class. This allows CDA to have an overall control among the entire prodution process with the purpose of centralization of the distributed agent systems.

As for the decentralization of MAS, the RA should be able to react to CDA and do decision making for its own field level control. Which means, CDA should only be informed about the status of the production process, while the exact processes should be done within RA and the material flow should be informed between RAs. The combination of centralization and decentralization will save the resource power consumed by CDA, while still maintains the management level control and decision making. The Algorithm 2 shows the logic of a RA. At the beginning, RA waits for the connection and capability check from CDA, after that first agent in the list of $sequenced\_agent$ will listen to the starting message from CDA, and other agents, will wait for the availability check and inform message of the last agent in the list. After executing all primitives of its own, the RA should check the availability and inform the starting point of next agent. Once all primitives in the list are done, the last agent should inform the CDA about the end of processing.

3.2 External

3.2.1 Overview (conceptual diagram)

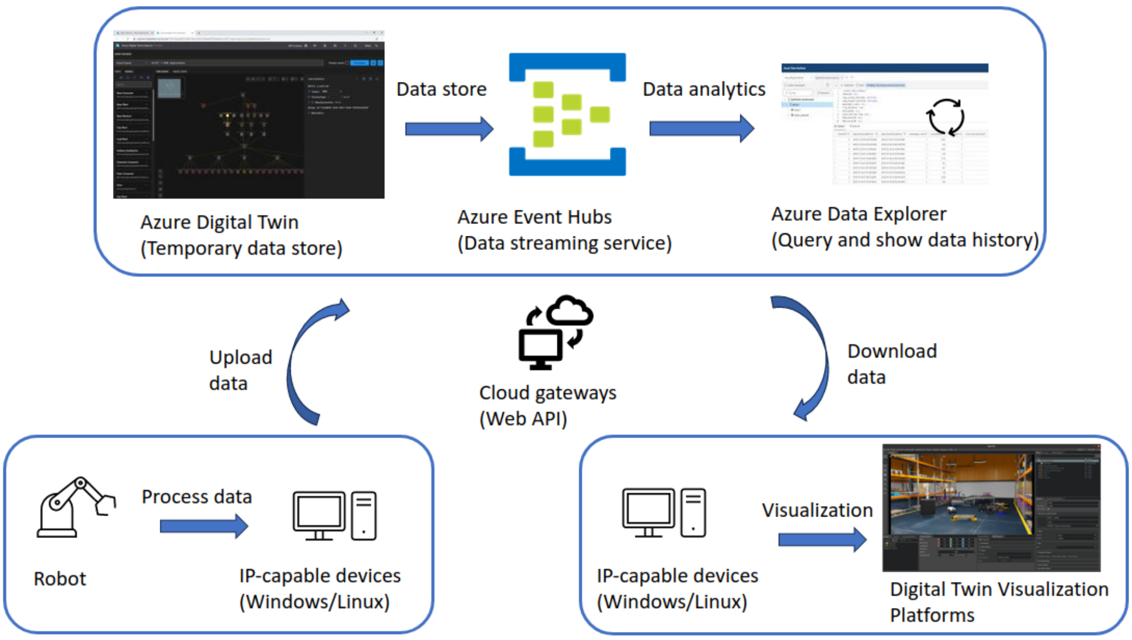

Figure 5: Conceptual diagram of MAS

The figure 5 shows the workflow of DT agent updating global DT. The process begins from the left hand side of the graph. The DT agent receives process data constantly from robots, including steady state data. After the separating the sticky packets from TCP socket, the process data will be parsed and uploaded one by one to the Microsoft (MS) Azure Digital Twin cloud. All the updates of the digital twin instances in cloud will be shown in MS Azure Digital Twin Explorer temporarily and stored in Azure Event Hubs concurrently. After that the history data could be ingested for further analysis in Azure Data Explorer. The Azure Digital Twin Explorer here serves as a monitor of robot status while the Azure Data Explorer as a data analysis tool. Once the remote data update is successful, the data will be downloaded to local host immediately and may be further used as inputs of any other visualization purposes. Within this upload and download cycle of data, the real time capability should also be maintained.

Algorithm 1 Pseudo-Code for Coordinator agent in MAS workflow

```
Input: custRequirement
Import WebSocket
Initialize agentID, centralServerIP
Class messageSender
    function send_and_receive(self, recipient, message, priority)
        Establish a WebSocket connection
        while recipientNotFound do
            Send prioritized messages and wait for response
    function send_capCheck_and_receive(self, recipient, primitive, priority)
        Establish a WebSocket connection
        while recipient not found do
            Send prioritized messages and wait for response
            Handle capability exceptions
    #similar to send_and_receive() but send an image
    function send_image_and_receive(self, recipient, image_path, priority)
        ...
    #similar to send_capCheck_and_receive() but raise availibility exceptions
    function send_availCheck_and_receive(self, recipient, primitive, priority)
        ...
EndClass
Class messageReceiver
    function receive_and_send(self, message, priority)
        Establish a WebSocket connection
        while message not received do
            Receive, process and send messages with priority
    #similar to receive_and_send() but receive and save an image
    function receive_image_and_send(self, message, priority)
        ...
EndClass
Class agentsAllocation
    function allocate_agents_with_seq_primitives(self, requirements)
        Find tasks from customer requirements
        Breakdown tasks into skills into primitives
        Create sequence lists for primitives agents allocation
EndClass
Main:
    Instantiate agentsAllocation, messageSender and messageReceiver
    repeat
        sequences = allocate_agents_with_seq_primitives(custRequirement)
    until all allocated agents connected
    repeat
        send_and_receive(agent, connectMsg, priority)
    until all allocated agents connected
    repeat
        send_capCheck_and_receive(agent, primitive, priority)
    until all allocated agents capable of all primitives
```

```
    repeat
        send_and_receive(agent, sequences, priority)
    until all allocated agents receive sequences
        send_availCheck_and_receive(1st_Agent, 1st_primitive, priority)
    repeat
        receive_and_send(responseMsg, priority)
    until informed by last agent with finish message
    End
```

Algorithm 2 Pseudo-Code for Resource agent in MAS workflow

```
Import necessary libraries
Initialize agent relevant variables
Class messageSender
    function send_and_receive_messages(self, recipient, message, priority)
        1. Establish a WebSocket connection
        2. Send and receive messages
        3. Handle exceptions
EndClass
Class messageReceiver
    function receive_and_send_messages(self, recipient, message, priority)
        1. Establish a WebSocket connection
        2. Receive, process and send messages
        3. Handle exceptions
EndClass
Main:
    Instantiate message sender and receiver
    Wait for connectivity check from Coordinator agent
     Receive capability check results from coordinator agent
    repeat
        if agent is first in sequence then
            wait for availability check from Coordinator agent
        else
            if previous agent is not self then
            wait receive availability check and inform message from last agent
            do execute corresponding primitive
            if agent is not last one AND next agent is not self then
                send availability check and inform message to next agent
                execute corresponding primitive
                inform Coordinator agent about completion
            if agent is last then
                execute corresponding primitive
                inform Coordinator agent about completion, send finish message
    End
```

It is worth mentioning that, the process data is produced by robot motion which makes the DT agent completely decoupled from the MAS. The advantages of the decoupling is that, even though there is a complete disorder of the MAS, the current robot states will be recorded and updated in remote, which is beneficial for data acquisition and monitoring.

3.2.2 Prerequisite

Network Time Protocol (NTP) Setup

In order to measure the delays between local host and cloud, the clock of both end must be synchronized. Due to the nature of computer systems, the software clock might drift away from the "true" time (absolute Coodinated Universal Time (UTC) time) due to various reasons like system load, hardware imperfections, or even temperature changes. Therefore, before the measurement, the software clock of local host should be synchronized with the global clock using NTP. The NTP setup of linux system are:

1. System update and upgrade.
2. Install NTP.
3. Add reliable global NTP servers/server pools to configure ntp.conf file.
4. Allow port 123/udp for or disable firewall.
5. Restart NTP and check NTP status.
6. Check synchronization status (e.g.: reach, delay, offset and jitter, etc).

Since the cloud system clock is already synchronized with UTC time, all the tests and calculations results are based on UTC time.

3.2.3 Pseudo-Code of Robot Control Program (RCP) and DT agent workflow in C and C#

Algorithm 3 Pseudo-Code of RCP in RCP-DTAgent workflow

Main:
  Instantiate robot
  Initialize start position
  Establish a TCP connection with DT agent
  Start robot motion and record robot state
  Send robot state to DTAgent
  End

Algorithm 4 Pseudo-Code of DT agent in RCP-DTAgent workflow

  Instantiate robot
  Initialize start position
  Establish a TCP connection with DT agent
  Start robot motion and record robot state
  Send robot state to DT agent
  End

3.2.4 MS Azure Digital Twin

Azure Digital twin database setup

4 Results and discussion

4.1 Test results of Websocket and Restful API

4.2 Test results of WS in diff. performance testing, worst case scenarios

4.3 priority tests of WS server in diff. performance testing

4.4 Test results of DTagents related to Azure Digital Twin

5 Conclusion and outlook

A Anhangname

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Bibliography

[1] Cruz Salazar, L. A., Ryashentseva, D., Lüder, A., and Vogel-Heuser, B. “Cyber-physical production systems architecture based on multi-agent’s design pattern—comparison of selected approaches mapping four agent patterns”. en. In: *The International Journal of Advanced Manufacturing Technology* 105.9 (Dec. 2019), pp. 4005–4034. ISSN: 0268-3768, 1433-3015. DOI: 10.1007/s00170-019-03800-4. URL: http://link.springer.com/10.1007/s00170-019-03800-4 (visited on 10/17/2023).

[2] Flumerfelt, S., Schwartz, K. G., Mavris, D., and Briceno, S. *Complex Systems Engineering: Theory and Practice*. Ed. by Schwartz, K. G., Mavris, D., Briceno, S., and Flumerfelt, S. _eprint: https://arc.aiaa.org/doi/pdf/10.2514/4.105654. Reston, VA: American Institute of Aeronautics and Astronautics, Inc., 2019. ISBN: 978-1-62410-564-7. DOI: 10.2514/4.105654. URL: https://arc.aiaa.org/doi/abs/10.2514/4.105654 (visited on 10/18/2023).

[3] Greengard, S. *Digital Twins Grow Up*. en. URL: https://cacm.acm.org/news/238642-digital-twins-grow-up/fulltext (visited on 10/18/2023).

[4] *Jacques Ferber: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. URL: https://www.jasss.org/4/2/reviews/rouchier.html (visited on 10/17/2023).

[5] Lauber, R. and Göhner, P. *Prozessautomatisierung 1*. de. Berlin, Heidelberg: Springer, 1999. ISBN: 978-3-540-65318-9 978-3-642-58446-6. DOI: 10.1007/978-3-642-58446-6. URL: http://link.springer.com/10.1007/978-3-642-58446-6 (visited on 10/17/2023).

[6] Ocker, F., Urban, C., Vogel-Heuser, B., and Diedrich, C. “Leveraging the Asset Administration Shell for Agent-Based Production Systems”. en. In: *IFAC-PapersOnLine* 54.1 (2021), pp. 837–844. ISSN: 24058963. DOI: 10.1016/j.ifacol.2021.08.186. URL: https://linkinghub.elsevier.com/retrieve/pii/S2405896321009563 (visited on 10/17/2023).

[7] Ota, J. “Multi-agent robot systems as distributed autonomous systems”. en. In: *Advanced Engineering Informatics* 20.1 (Jan. 2006), pp. 59–70. ISSN: 14740346. DOI: 10.1016/j.aei.2005.06.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S1474034605000509 (visited on 10/17/2023).

[8] Vogel-Heuser, B., Seitz, M., Cruz Salazar, L. A., Gehlhoff, F., Dogan, A., and Fay, A. “Multi-agent systems to enable Industry 4.0”. en. In: *at - Automatisierungstechnik* 68.6 (June 2020), pp. 445–458. ISSN: 2196-677X, 0178-2312. DOI: 10.1515/auto-2020-0004. URL: https://www.degruyter.com/document/doi/10.1515/auto-2020-0004/html (visited on 10/17/2023).

[9] Wagner, T. “Agentenunterstütztes Engineering von Automatisierungsanlagen”. de. Accepted: 2008-05-19 Journal Abbreviation: Agent-supported engineering of industrial plants. doctoralThesis. 2008. DOI: 10.18419/opus-2630. URL: http://elib.uni-stuttgart.de/handle/11682/2647 (visited on 10/17/2023).

[10] Wannagat, A and Vogel-Heuser, B. “Agent oriented software-development for networked embedded systems with real time and dependability requirements the domain of automation”. en. In: ().

[11] Wannagat, A. "Entwicklung und Evaluation agentenorientierter Automatisierungssysteme zur Erhöhung der Flexibilität und Zuverlässigkeit von Produktionsanlagen". PhD thesis. Technische Universität München, 2010. URL: https://mediatum.ub.tum.de/958364 (visited on 10/17/2023).

List of Abbreviations

AMS	Agent Management System
CA	Communication Agent
CDA	Coordinator Agent
CPS	Cyber Physical System
DAI	Distributed Artificial Intelligence
DB	Database
DSL	Domain Specific Language
DT	Digital Twin
HTTP	Hypertext Transfer Protocol
LBAM	Professorship Laser-based Additive Manufacturing
MAS	Multi-Agent System
MFS	Material Flow System
MQTT	Message Queuing Telemetry Transport
MS	Microsoft
NTP	Network Time Protocol
PLM	Product Lifecycle Management
RA	Resource Agent
RAs	Resource Agents
RCP	Robot Control Program
TCP	Transmission Control Protocol
UDP	User Diagram Protocol
UTC	Coodinated Universal Time

Disclaimer

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Garching, November 15, 2023

(Signature)